How to create Visual Basic macros by using Excel Solver in Excel 97

Article ID : 843304
Last Review : November 4, 2004
Revision : 1.0

SUMMARY

This article describes how to use Microsoft Excel Solver in Microsoft Excel 97 to create Microsoft Visual Basic macros. Microsoft Excel Solver is a Microsoft Excel add-in.

Additionally, this article contains information about how to create macros, how to design a macro, and how to work with constraints of a macro. This article also discusses the algorithm and methods that are used by Microsoft Excel Solver. The following list gives all the topics discussed in the article.

INTRODUCTION

This article contains information about Microsoft Excel Solver.

MORE INFORMATION

Description of the Microsoft Excel Solver

Microsoft Excel Solver is a Microsoft Excel add-in. Microsoft Excel Solver helps you to determine the optimum value for a formula in a particular target cell on a Microsoft Excel worksheet. Microsoft Excel Solver adjusts the values of other cells that are related to the target cell by using an equation. After you construct an equation and define a set of parameters or constraints for the variables in the equation, Microsoft Excel Solver tries various solutions to arrive at an answer that satisfies all constraints. Microsoft Excel Solver uses the following elements to "solve" an equation:

- **Target cell** - The target cell is the objective. It is the cell in the worksheet model that will be minimized, maximized, or set to a certain value.
- **Changing cells** - Changing cells are the decision variables. These cells affect the value of the target cell. These cells are changed by Microsoft Excel Solver to find the optimum solution for the target cell.
- **Constraints** - Constraints are restrictions on the contents of cells. For example, one cell in a worksheet model may be restricted to integer values, while another cell may be restricted to being less than a given value.

You can automate the creation and the manipulation of Microsoft Excel Solver models by using a Microsoft Visual Basic for Applications (VBA) macro. This article describes how to use the VBA macro language to use the Microsoft Excel Solver functions in Microsoft Excel 97. This article assumes that you are familiar with the VBA language and the Microsoft Visual Basic Editor for Microsoft Excel 97. The examples that are used in this article are available for download at the following Microsoft Web site:

http://download.microsoft.com/download/excel97win/solverex/1.0/WIN98Me/EN-US/SolverEx.exe

Note You can also use the macros and the examples that are described in this article in Microsoft Excel versions 5.0 and 7.0.

back to the top

How to use the Microsoft Excel Solver functions in a VBA macro

To use the Microsoft Excel Solver add-in functions in a VBA macro, you must reference the add-in from the VBA project of the workbook that contains the macros. If you do not reference the Microsoft Excel Solver add-in, you will receive the following compile error when you try to run the macro:

Compile Error: Sub or Function not defined.

To reference the Microsoft Excel Solver add-in for macros in your workbook, use the following steps:

1. Open your workbook.
2. On the **Tools** menu, point to **Macro**, and then click **Visual Basic Editor**.
3. On the **Tools** menu, click **References**.
4. In the **Available References** list, click to select the **Solver.xls** check box, and then click **OK**.

   Note If you do not see Solver.xls in the **Available References** list, click **Browse**. In the **Add Reference** dialog box, locate and select the Solver.xla file, and then click **Open**. The Solver.xla file is typically found in the C:\Program Files\Microsoft Office\Office\Library\Solver subfolder.

You are now ready to use the Microsoft Excel Solver functions in a VBA macro.

back to the top

How to design a VBA macro that creates and solves a simple Microsoft Excel Solver model

Although Microsoft Excel Solver offers many functions, the following three functions are fundamental to creating and to solving a model:

- The **SolverOK** function
- The **SolverSolve** function
- The **SolverFinish** function

The SolverOK function

The **SolverOK** function defines a basic Microsoft Excel Solver model. The **SolverOK** function is generally the first function that you will use to build your Microsoft Excel Solver model. The **SolverOK** function is equivalent to clicking **Solver** on the **Tools** menu, and then specifying the options that are in the **Solver Parameters** dialog box. The following is the syntax for the **SolverOK** function:

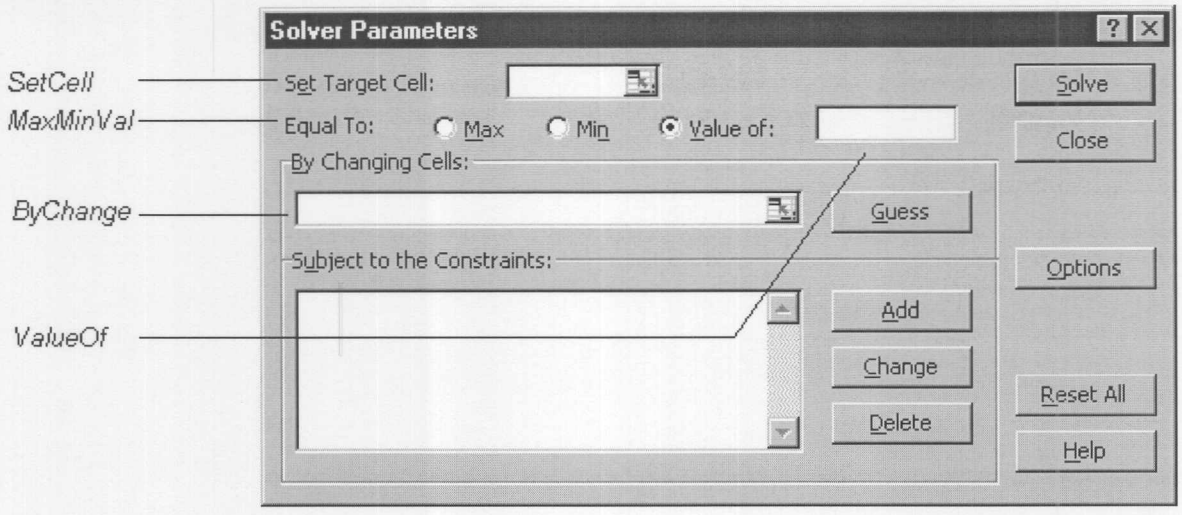
SolverOK(SetCell, MaxMinVal, ValueOf, ByChange)

The following information describes the syntax for the **SolverOK** function:

- **SetCell** specifies the target cell.
- **MaxMinVal** corresponds to whether you want to solve the target cell for a maximum value (1), a minimum value (2), or a specific value (3).
- **ValueOf** specifies the value to which the target cell is matched. If you set **MaxMinVal** to 3, you must specify this argument. If you set **MaxMinVal** to 1 or 2, you can omit this argument.
- **ByChange** specifies the cell or range of cells that will be changed.

Figure 1 associates the arguments for the **SolverOK** function with the parameters in the **Solver Parameters** dialog box.

Figure 1. Parameters that are associated with the ***SolverOK*** arguments



The SolverSolve function

The **SolverSolve** function solves the model using the parameters that you specified with the **SolverOK** function. Executing the **SolverSolve** function is equivalent to clicking **Solve** in the **Solver Parameters** dialog box. The following is the syntax for the **SolverSolve** function:

SolverSolve(UserFinish, ShowRef)

The following information describes the syntax for the **SolverSolve** function:

- **UserFinish** indicates whether you want the user to finish solving the model.

  To return the results without displaying the **Solver Results** dialog box, set this argument to TRUE. To return the results and display the **Solver Results** dialog box, set this argument to FALSE
- **ShowRef** identifies the macro that is called when Microsoft Excel Solver returns an intermediate solution.

  The ***ShowRef*** argument should be used only when TRUE is passed to the ***StepThru*** argument of the **SolverOptions** function.

The SolverFinish function

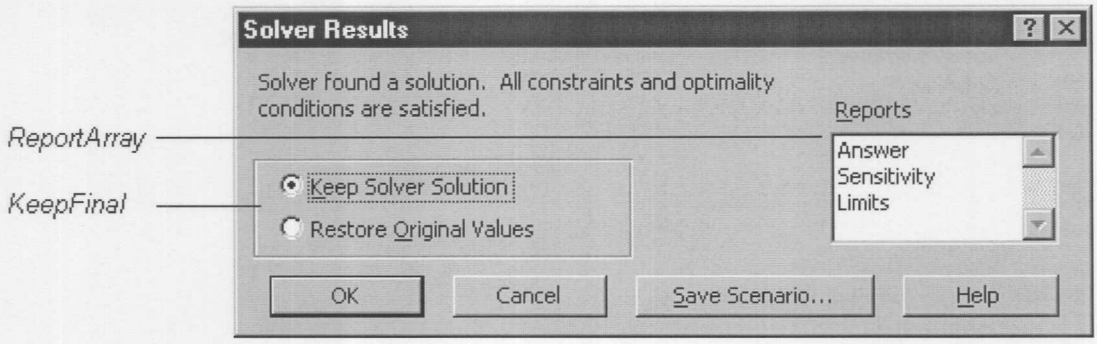
The **SolverFinish** function indicates what to do with the results, and what kind of report to create after the solution process is completed. The following is the syntax for the **SolverFinish** function:

SolverFinish (KeepFinal, ReportArray)

The following information describes the syntax for the **SolverFinish** function:

- **KeepFinal** indicates what to do with the final results. If **KeepFinal** is 1, the final solution values are kept in the changing cells, replacing the values. If **KeepFinal** is 2, the final solution values are discarded, and the former values are restored.
- **ReportArray** specifies an array which indicates the type of report Microsoft Excel will create when the solution is reached. If **ReportArray** is set to 1, Microsoft Excel creates an Answer Report. If set to 2, Microsoft Excel creates a Sensitivity Report, and if set to 3 Microsoft Excel creates a Limits Report. For more information about these reports, see "How to generate reports for solutions" section.

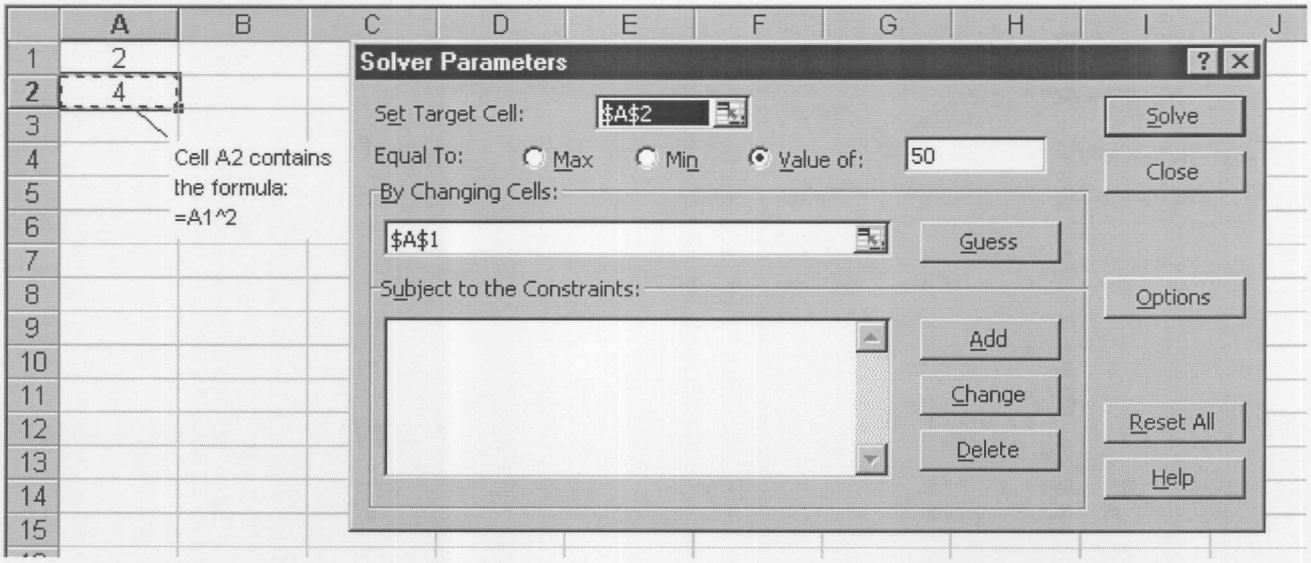
Figure 2. Microsoft Excel Solver results options that are associated with ***SolverFinish*** arguments



This article describes how to create a simple Microsoft Excel Solver model interactively. The first step is to create your worksheet for the model. The worksheet will contain some data cells and at least one cell that contains a formula. This formula depends on the other cells in the worksheet. After you set up your worksheet, click **Solver** on the **Tools** menu. In the **Solver Parameters** dialog box, specify the target cell, the value that you are solving for, the range of cells that will be changed, and the constraints. Click **Solve** to start the solution process. After Microsoft Excel Solver has found a solution, the results appear in your worksheet, and the Microsoft Excel Solver displays a message box that prompts you if you want to keep the final results or if you want to discard them. When you click one of these options, Microsoft Excel Solver finishes.

Figure 3 illustrates a simple model that you can create by using these steps.

Figure 3. A simple model: The Square Root model



In this example, change cell A1, which contains the formula, =A1^2, to a value that will make cell A2 equal to a value of 50. In other words, find the square root of 50. There are no constraints in the Square Root model. The **Find_Square_Root** macro accomplishes the following tasks:

- It sets up a model that will solve the value of cell A2 for a value of 50 by changing the value of cell A1.
- It solves the model.
- It saves the final results to the worksheet without displaying the **Solver Results** dialog box.

This simple macro creates a Microsoft Excel Solver model and solves it without any user intervention. The following code describes the **Find_Square_Root** macro:

```
Sub Find_Square_Root()

' Set up the parameters for the model.
' Set the target cell A2 to a value of 50 by changing cell A1.
SolverOK SetCell:=Range("A2"), MaxMinVal:=3, ValueOf:=50, _
     ByChange:=Range("A1")
```

```
' Solve the model but do not display the Solver Results dialog box.
SolverSolve UserFinish:=True

' Finish and keep the final results.
SolverFinish KeepFinal:=1

End Sub
```

The **Find_Square_Root2** macro, is a modified version of the **Find_Square_Root** macro. If you use the **InputBox** function, the **Find_Square_Root2** macro prompts you for the value that you want to solve for the target cell. After you input a value, the **Find_Square_Root2** macro sets this parameter as the value of the **SolverOK*valueof*** argument, solves the problem, saves the results in the variable square root, and then discards the solution and restores the value in the worksheet to its original state. Basically, the **Find_Square_Root2** macro illustrates how you can save the results in one or more variables and then restore the changing cells to their original value.

The following code describes the **Find_Square_Root2** macro:

```
Sub Find_Square_Root2()

Dim val
Dim sqroot

' Request the value for which you want to obtain the square root.
val = Application.InputBox( _
     prompt:="Please enter the value for which you want " & _
     "to find the square root:", Type:=1)

' Set up the parameters for the model.
SolverOK SetCell:=Range("A2"), MaxMinVal:=3, ValueOf:=val, _
     ByChange:=Range("A1")

' Do not display the Solver Results dialog box.
SolverSolve UserFinish:=True

' Save the value of cell A1 (the changing cell) before you discard
' the results.
sqroot = Range("a1")

' Finish and discard the results.
SolverFinish KeepFinal:=2

' Show the result in a message box.
MsgBox "The square root of " & val & " is " & Format(sqroot, "0.00")

End Sub
```

back to the top

How to generate reports for solutions

Microsoft Excel Solver offers several types of reports that describe how the results changed and how close the constraints came to their critical values. Each report is put on a separate worksheet in your workbook. These following are the types of reports that the Microsoft Excel Solver offers:

- **Answer Report** - The Answer Report lists the target cell and the changing cells with their corresponding original and final values, constraints, and information about the constraints.
- **Sensitivity Report** - The Sensitivity Report provides information about how sensitive the solution is to small changes in the formula for the target cell.
- **Limits Report** - The Limits Report lists the target cell and the changing cells with their respective values, the lower and upper limits, and the target values.

To create reports for your models, specify an array of values for the ***ReportArray*** argument of the **SolverFinish** function. For more information about the ***ReportArray*** argument, see the "SolverFinish (KeepFinal, ReportArray) " section. For example, if you want to generate a Limits Report for the model that the **Find_Square_Root2** macro creates and solves, modify the **SolverFinish** function in the macro so that it looks similar to the following example code:

```
SolverFinish KeepFinal:=2, ReportArray:= Array(3)
```

To generate multiple reports, modify the **SolverFinish** function so that it looks similar to the following sample code:

```
SolverFinish KeepFinal:=2, ReportArray:= Array(1,2)
```

back to the top

How to use the Microsoft ExcelSolver functions in a looping macro

In many situations, it is a good idea to have Microsoft Excel Solver solve the target cell for multiple values. You can generally accomplish this by using one of the looping structures that are available with VBA.

The **Create_Square_Root_Table** macro demonstrates how Microsoft Excel Solver functions in a looping macro. The **Create_Square_Root_Table** macro creates a table in a new worksheet. It inserts the numbers one through ten and the corresponding square root of each number. The **Create_Square_Root_Table** macro creates the table using a **For** loop to iterate through the numbers 1 through 10 and to solve the target cell in the Square Root model for a value that matches the number of the iteration. The following code describes the **Create_Square_Root_Table** macro:

```
Sub Create_Square_Root_Table()

' Add a new worksheet to the workbook.
Set w = Worksheets.Add

' Put the value 2 in cell C1 and the formula =C1^2 in cell C2.
w.Range("C1").Value = 2
w.Range("C2").Formula = "=C1^2"

' A loop that will make 10 iterations, starting with the number 1,
' and finishing at the number 10.
For i = 1 To 10

    ' Set the Solver parameters that indicate that Solver should
    ' solve the cell C2 for the value of i (where i is the number
    ' of the iteration) by changing cell C1.
    SolverOk SetCell:=Range("C2"), ByChange:=Range("C1"), _
        MaxMinVal:=3, ValueOf:=i

    ' Do not display the Solver Results dialog box.
    SolverSolve UserFinish:=True

    ' Save the value of i in column A and the results of the
    ' changing cell in column B.
    w.Cells(i, 1) = i
    w.Cells(i, 2) = Range("C1")

    ' Finish and discard the final results.
    SolverFinish KeepFinal:=2

Next

' Clear the range C1:C2
w.Range("C1:C2").Clear

End Sub
```

The **Create_Square_Root_Table** macro generates the table illustrated in Figure 4.

Figure 4. Output that is generated by the **Create_Square_Root_Table** macro

	A	B
1	1	1
2	2	1.414213
3	3	1.73205
4	4	2
5	5	2.236068
6	6	2.449489
7	7	2.645752
8	8	2.828427
9	9	3
10	10	3.162278

back to the top

How to work with constraints

A constraint is a restriction on the contents of one or more cells. A model can have one or multiple constraints. The constraint set is a set of inequalities or a set of equalities that remove certain combinations of values for the decision variables from the solution. For example, a constraint may require that one cell be greater than zero and that another cell contain only an integer value.

The Square Root model that we have discussed up to this point is a simple model that does not contain any constraints. Figure 5 illustrates a model that uses constraints. The purpose of this model is to find the optimal combination of products for maximum profit.

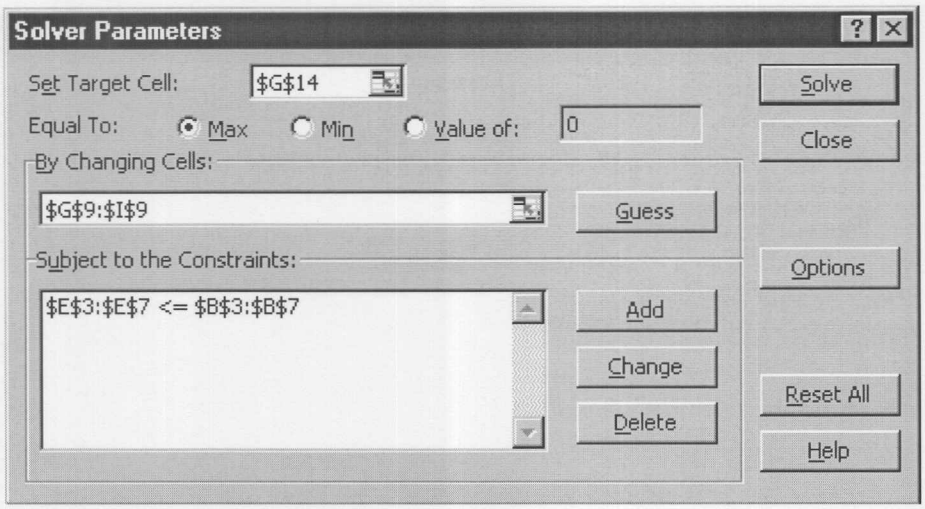
Figure 5. Product mix with diminishing profit margin

	A	B	C	D	E	F	G	H	I
1	Parts Inventory								
2	Name	On Hand	Ordered	Projected	Used		TV Set	Stereo	Speaker
3	Chassis	450	55	505	200		1	1	0
4	Picture Tube	250	20	270	100		1	0	0
5	Speaker Cone	800	100	900	500		2	2	1
6	Power Supply	450	60	510	200		1	1	0
7	Electronics	600	80	680	400		2	1	1
8									
9				Units to Build			100	100	100
10				Profit Per Unit			$75	$50	$35
11				Profit By Product			$4,732	$3,155	$2,208
12									
13									
14				Total Profit			$10,095		

For example, if a company manufactures TVs, stereos, and speakers, and it uses a common parts inventory of power supplies, speaker cones, and so on. The parts are in limited supply. Your goal is to determine the most profitable mix of products to build. Your profit per unit decreases with volume because additional price incentives are required to load the distribution channel. The diminishing returns exponent is 0.9. This exponent is used to calculate the profit by product in the range G11:I11.

Your objective is to find the maximum profit (cell G14). The values that you will change to find the maximum profit are the number of units that you build. The range G9:G11 represents the changing cells in this model. Your only constraint is that the number of parts you use cannot exceed the number of parts you have on hand. With Microsoft Excel Solver, this constraint appears as E3:E7<=B3:B7. If you were to build this Microsoft Excel Solver model interactively, the Microsoft Excel Solver parameters would look similar to those that are in Figure 6.

Figure 6. Microsoft Excel Solver parameters for the product mix with Diminishing Profit Margin model



To create and solve the Product Mix with Diminishing Profit Margin model, you will use a new function, the **SolverAdd** function, in addition to the Microsoft Excel Solver VBA functions that were described earlier. The **SolverAdd** function adds the constraint to the model. Executing the **SolverAdd** function is equivalent to clicking the **Add** button in the **Solver Parameters** dialog box. The **SolverAdd** function has the following syntax:

SolverAdd (CellRef, Relation, FormulaText)

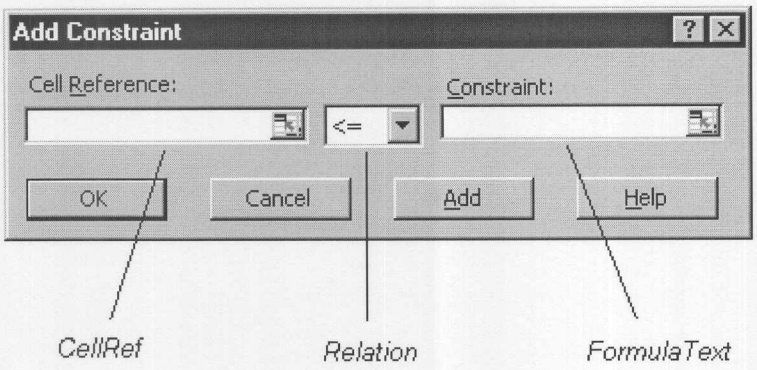
The following information describes the syntax for the **SolverAdd** function:

- **CellRef** references one or more cells that form the left side of the constraint.
- **Relation** is the arithmetic relationship between the left and the right sides of a constraint.
- **Relation** can be a value between 1 and 5 as in the following example:
  - The value 1 is less than or equal to (<=).
  - The vaue 2 is equal to (=).
  - The value 3 is greater than or equal to (>=).
  - The value 4 is an integer.
  - The value 5 is the binary (a value of zero or one).
- **FormulaText** references one or more cells that form the right side of the constraint.**

When you specify a range of cells for the ***FormulaText* argument of the **SolverAdd** function, note whether the reference is relative or absolute. Generally, you must specify an absolute reference for the ***FormulaText*** argument. However, if you do specify relative references for the ***FormulaText*** argument, realize that the reference will be relative to the target cell and not the active cell.

Note In Microsoft Excel, versions 5.0 and 7.0, use the R1C1 notation when you specify a cell or a range of cells with the ***FormulaText*** argument. In contrast, in Microsoft Excel 97, use the A1-style notation to specify the ***FormulaText*** argument.

Figure 7. Fields that are associated with the ***SolverAdd*** arguments



The **Maximum_Profit** macro that generates a model for the Product Mix with Diminishing Returns model. This macro executes the following functions or arguments:

- The **SolverOK** function sets up the target cell for a maximum value and specifies the cells to change.
- The **SolverAdd** function adds the constraint to the model.
- The **SolverSolve** function finds a solution without displaying the **Solver Results** dialog box.
- The **SolverFinish** function returns the final results to the worksheet.

The following code describes the for **Maximum_Profit** macro:

```
Sub Maximum_Profit()

' Set up the parameters for the model.
' Determine the maximum value for the sum of profits in cell G14
' by changing the number of units to build in cells G9:I9.
Solverok setcell:=Range("G14"), maxminval:=1, _
    bychange:=Range("G9:I9")

' Add the constraint for the model. The only constraint is that the
' number of parts used does not exceed the parts on hand--
' E3:E7<=B3:B7
SolverAdd CellRef:=Range("E3:E7"), Relation:=1, _
    FormulaText:="$B$3:$B$7"

' Do not display the Solver Results dialog box.
SolverSolve UserFinish:=True

' Finish and keep the final results.
SolverFinish KeepFinal:=1

End Sub
```

Note In Microsoft Excel, versions 5.0 and 7.0, use the R1C1 notation when you specify a cell or range of cells with the ***FormulaText*** argument. In contrast, in Microsoft Excel 97, use the A1-style notation to specify the ***FormulaText*** argument.

When you run the **Maximum_Profit** macro, Microsoft Excel Solver will find a solution of building 160 TV sets, 200 stereos, and 80 speakers for a maximum profit of $14,917 dollars.

back to the top

How to change and delete constraints

Constraints in your model can be programmatically changed or deleted. Constraints are identified by their ***CellRef*** and ***Relation*** arguments.

To programmatically change an existing constraint, use the **SolverChange** function. The following is the syntax for the **SolverChange** function:

SolverChange (CellRef, Relation, FormulaText)

Note that the arguments for the **SolverChange** function are the same to those that you use with the **SolverAdd** function.

If you want to change the constraint in the Product Mix with Diminishing Returns model, you would use the **SolverChange** function. For example, currently the constraint that is specified is that the number of parts used is less than or equal to the number of parts on hand (E3:E7 <= B3:B7). If you want to change this constraint so that the number of parts used is less than or equal to the number of parts projected (number of parts on hand plus number of parts ordered). This new constraint would look like E3:E7 <= D3:D7. The following macro would change the existing constraint E3:E7<=B3:B7 to E3:E7 <= D3:D7 and solve for a solution.

The following code describes the **Change_Constraint_and_Solve** macro:

```
Sub Change_Constraint_and_Solve()

' Change the constraint.
SolverChange CellRef:=Range("E3:E7"), Relation:=1, _
   FormulaText:="$D$3:$D$7"

' Return the results and display the Solver Results dialog box.
SolverSolve UserFinish:=False

End Sub
```

Because constraints are identified by the ***CellRef*** and ***Relation*** arguments, you can only change the ***FormulaText*** argument for the constraint by using the **SolverChange** function. If the **CellRef** and the **Relation** values do not match an existing constraint, you must delete the constraint and then add the modified constraint. To delete a constraint, use the **SolverDelete** function. The following is the syntax for the **SolverDelete** function:

SolverDelete (CellRef, Relation, FormulaText)

Note that the arguments for the **SolverDelete** function are the same to those you use with the **SolverAdd** and the **SolverChange** functions.

The following macro illustrates how to delete and add a constraint. In this example, the **Change_Constraint_and_Solve2** macro removes the constraint E3:E7<=B3:B7 from the Product Mix with Diminishing Returns model and adds a new constraint. The new constraint is just a modification of the original constraint, where the left and right sides of the constraint are reversed.

The following code describes the **Change_Constraint_and_Solve2** macro:

```
Sub Change_Constraint_and_Solve2()

' Reverse the left and right sides of the constraint...
' Delete the constraint E3:E7<=B3:B7 and add the
' constraint B3:B7>=E3:E7.
SolverDelete CellRef:=Range("E3:E7"), Relation:=1, _
    FormulaText:="$B$3:$B$7"
SolverAdd CellRef:=Range("B3:B7"), Relation:=3, _
    FormulaText:="$E$3:$E$7"

' Return the results and display the Solver Results dialog box.
SolverSolve UserFinish:=False

End Sub
```

Note In Microsoft Excel, versions 5.0 and 7.0, use the R1C1 notation when you specify a cell or range of cells with the ***FormulaText*** argument. In contrast, in Microsoft Excel 97, use the A1-style notation to specify the ***FormulaText*** argument.

back to the top

How to load and save your models

When you save your workbook, the last parameters that you specified in the **Solver Parameters** dialog box are saved with the workbook. Therefore, when you open the workbook, the parameters are the same as when you last saved the workbook.

You can define more than one problem for a worksheet. Each problem is made up of cells and constraints that you

enter in the **Solver Parameter** and the **Solver Options** dialog boxes. Because only the last problem is saved with the worksheet, you will lose all other problems unless you explicitly save them. To save them, click **Save Model** in the **Solver Options** dialog box. Similarly, when you want to restore the previously saved parameters, click **Load Model** in the **Solver Options** dialog box.

Solver models are stored in a range of cells on a worksheet. The first cell in the range contains the formula for the target cell. The second cell in the range contains the formula that identifies the changing cells in the model. The last cell in the range contains an array that represents the options set in the **Solver Options** dialog box. The cells between the second cell and the last cell contain the formulas that represent the constraints in the model.

Figure 8 illustrates a model for employee scheduling. Assume that you work for a small manufacturer. This table shows each employee's hourly rate of pay, the number of hours they are scheduled, and a projected number of units each employee can produce in one hour. Your goal is to meet a specific quota for the number of units produced while minimizing the cost of labor.

Figure 8. Employee Scheduling mode
l

	A	B	C	D	E	F	G	H
1	**Employee**	**Hourly Rate**	**Hours Worked**	**Regular Pay**	**Overtime Pay**	**Units Per Hour**	**Units Per Week**	
2	Brown	$8.25	40	$330.00	$0.00	13	520	
3	Thomas	$9.00	40	$360.00	$0.00	15	600	
4	Adams	$8.75	40	$350.00	$0.00	14	560	
5	Jones	$7.85	40	$314.00	$0.00	13	520	
6	Smith	$8.25	40	$330.00	$0.00	12	480	
7	Johnson	$9.25	40	$370.00	$0.00	14	560	
8	Thompson	$8.75	40	$350.00	$0.00	14	560	
9								
10								
11			**Cost of Labor:**			**Units Produced:**		
12				$2,404.00			3800	
13								

Two additional factors (or constraints) that you must consider are the minimum/maximum number of hours any one employee can work and the number of units that you intend to produce. If for a specified week, you need to produce 3975 units and you want each employee to work between 30 and 45 hours, the Microsoft Excel Solver parameters would look similar to those outlined in the following table:

Parameter	**Cell Range**	**Description**
Target cell	$D$12	Cost of labor.
Changing cells	$C$2:$C$8	Hours worked per employee.
Constraints	$C$2:$C$8 <= 45	Maximum hours per employee is 45.
	$C$2:$C$8 >= 30	Minimum hours per employee is 35.
	$G$12 = 3975	Number of units is 3975.

Your goals are to solve for optimal labor cost on a weekly basis, to save each model weekly, and to be able to load any weekly model when you need it.

In a macro, the Microsoft Excel Solver parameters for a model can be saved and loaded by using the **SolverSave** and the **SolverLoad** functions respectively. The **SolverSave** and the **SolverLoad** functions have the following syntax:

SolverSave (SaveArea)

SolverLoad (LoadArea)

The **SolverSave** and the **SolverLoad** functions each have only one argument, ***SaveArea*** and the ***LoadArea*** arguments respectively. These arguments specify a range on a worksheet where the model information is stored.

The following **New_Employee_Schedule** macro, demonstrates how to create, to solve, and to save a model based on user input. The user is asked to supply the date of the model, the number of units to produce, and the minimum and maximum number of hours per employee. These data is then used to create the model. The model is solved and then saved with the user input.

The following code describes the **New_Employee_Schedule** macro:

```
Sub New_Employee_Schedule()

' Prompt the user for the date of the model, the units to produce,
' and the maximum and minimum number of hours per employee.
ModelDate = Application.InputBox( _
   Prompt:="Date of Model:", Type:=2)
Units = Application.InputBox( _
   Prompt:="Projected Number of Units:", Type:=1)
MaxHrs = Application.InputBox( _
   Prompt:="Maximum Number of Hours Per Employee:", Type:=1)
MinHrs = Application.InputBox( _
   Prompt:="Minimum Number of Hours Per Employee:", Type:=1)

' Clear any previous Solver settings.
SolverReset

' Set the target cell, D12, to a minimum value by changing
' the range, C2:C8.
SolverOk SetCell:=Range("$D$12"), MaxMinVal:=2, _
   ByChange:=Range("C2:C8")

' Add the constraint that number of hours worked <= MaxHrs.
SolverAdd CellRef:=Range("C2:C8"), Relation:=1, FormulaText:=MaxHrs

' Add the constraint that number of hours worked >=MinHrs.
SolverAdd CellRef:=Range("C2:C8"), Relation:=3, FormulaText:=MinHrs

' Add the constraint that number of units produced = Units.
SolverAdd CellRef:=Range("G12"), Relation:=2, FormulaText:=Units

' Solve the model and keep the final results.
SolverSolve UserFinish:=True
SolverFinish KeepFinal:=1

' Save the input values for ModelDate, MaxHrs, MinHrs, and Units
' in columns I:L.
Set ModelRange = Range("I2:R2").CurrentRegion.Offset( _
   Range("I2:R2").CurrentRegion.Rows.Count).Resize(1, 1)
ModelRange.Resize(1, 4) = Array("'" & Format(ModelDate, "m/d/yy"), _
   Units, MaxHrs, MinHrs)

' Save the model parameters to the range M:R in the worksheet.
SolverSave SaveArea:=ModelRange.Offset(, 4).Resize(1, 6)

End Sub
```

Note In Microsoft Excel, versions 5.0 and 7.0, use the R1C1 notation when you specify a cell or range of cells with the ***FormulaText*** argument. In contrast, in Microsoft Excel 97, use the A1-style notation to specify the ***FormulaText*** argument.

Figure 9 illustrates how the saved model information appears on the worksheet.

Figure 9. Model information that is saved by the New_Employee_Schedule macro

	I	J	K	L	M	N	O	P	Q	R
1	*Saved Models*									
2	*Date*	*Units*	*Max Hrs*	*Min Hrs*	*Model Parameters*					
3	*12/1/96*	*4000*	*60*	*38*	*$2,530.30*	*7*	*TRUE*	*TRUE*	*TRUE*	*100*
4	*12/8/96*	*3200*	*48*	*30*	*$2,530.30*	*7*	*TRUE*	*TRUE*	*FALSE*	*100*
5	*12/15/96*	*3500*	*45*	*35*	*$2,530.30*	*7*	*TRUE*	*TRUE*	*FALSE*	*100*
6										

The **New_Employee_Schedule** macro saves each new model to the worksheet. The **Load_Employee_Schedule** macro can load one of these saved models. The macro prompts the user for the model to load and then searches column I for the model date. If the model date is found, the **Load_Employee_Schedule** macro loads the corresponding model, solves it, and then keeps the final results.

The following code describes the **New_Employee_Schedule**macro:

```
Sub Load_Employee_Schedule()

' Prompt for the date of the model.
ModelDate = Application.InputBox( _
   Prompt:="Date of Model to Load:", Type:=2)

' Locate the date in column I.
Set DateRange = Range("I2").CurrentRegion.Resize(, 1)
r = Application.Match(ModelDate, DateRange, 0)

If IsError(r) Then
    ' Display a message if the model date is not found
    MsgBox "Cannot find a model with the date " & ModelDate
Else
    ' If the model date is found, load the model into Solver,
    ' solve the model, and keep the final results.
    SolverLoad LoadArea:=DateRange.Offset(r - 1, 4).Resize(1, 6)
    SolverSolve UserFinish:=True
    SolverFinish KeepFinal:=1
End If

End Sub
```

The **New_Employee_Schedule** macro introduces the **SolverReset** function. The **SolverReset** function can be used to delete all cell selections and constraints in the **Solver Parameters** dialog box and to reset all settings in The **SolverReset** function has no arguments.

back to the top

How to find more information about Microsoft Excel Solver

The following resources provide information about how to use the Microsoft Excel Solver add-in.

- For help with specific solver messages, see Frontline Systems.
- For hints on building readable, manageable models, see Frontline Systems.
- For additional information about the Solver limits for constraints and, click the following article number to view the article in the Microsoft Knowledge Base:

  75714 Solver limits for constraints

- For several examples that use the Microsoft Excel Solver add-in in Microsoft Excel, see the Solvsamp.xls sample file.
- The following is the default location of the sample file that is included with Microsoft Excel 97:

  \Program Files\Microsoft Office\Office\Examples\Solver\SolvSamp.xls

- The following is the default location of the sample file that is included with Microsoft Excel 7.0:

\MSOffice\Excel\Examples\Solver\SolvSamp.xls

- The following is the default location of the sample file that is included with Microsoft Excel 5.0:

  \Excel\Examples\Solver\SolvSamp.xls

back to the top

How to learn more about the algorithm and methods that are used by Microsoft Excel Solver

Microsoft Excel Solver uses the Generalized Reduced Gradient (GRG2) nonlinear optimization code that was developed by Leon Lasdon, University of Texas at Austin, and Allan Waren, Cleveland State University.

For additional information about the algorithm used by Microsoft Excel Solver, click the following article number to view the article in the Microsoft Knowledge Base:

82890 Solver uses generalized reduced

Linear and integer problems use the simplex method with bounds on the variables, and the branch-and-bound method, implemented by John Watson and Dan Fylstra, Frontline Systems, Inc. For more information about the internal solution process used by Solver, contact:

Frontline Systems, Inc.
P.O. Box 4288
Incline Village, NV 89450-4288
(702) 831-0300
Web site: http://www.frontsys.com
Electronic mail: info@frontsys.com

Note The Microsoft Excel Solver add-in that is discussed in this article is provided "as is" and we do not guarantee that it can be used in all situations. Although Microsoft Support Professionals can help with the installation and existing functionality of this add-in, they will not modify the add-in to provide new functionality.

NO WARRANTY. The software is provided "as-is," without warranty of any kind, and any use of this software product is at your own risk.

back to the top

APPLIES TO

- Microsoft Excel 97 Standard Edition
- Microsoft Visual Basic for Applications 6.0

Keywords: kberrmsg kbhowto kbinfo kbmacroexample kbaddin kbprogramming kbvba KB843304